Techniques for reducing and bounding OpenMP dynamic memory

Adrian Munera*†, Sara Royuela*†, Eduardo Quiñones*
*Barcelona Supercomputing Center, Barcelona, Spain
†Universitat Politècnica de Catalunya, Barcelona, Spain
E-mail: {adrian.munera, sara.royuela, eduardo.quinones}@bsc.es

***Abstract*—OpenMP offers a tasking model very convenient to develop critical real-time parallel applications by virtue of its time predictability. However, current implementations make an intensive use of dynamic memory to efficiently manage the parallel execution. This jeopardizes the qualification process and limits the use of OpenMP in architectures with limited amount of memory. This work introduces an OpenMP framework that statically allocates the data structures needed to efficiently manage parallel execution in OpenMP programs. We achieve the same performance than current implementations, while bounding and reducing the dynamic memory requirements at runtime.**

***Keywords*—OpenMP, dynamic memory, critical-systems.**

I. Introduction

Current embedded architectures for critical real-time systems such as the Kalray MPPA [1], and the TI Keystone II [2] deliver levels of parallelism that can efficiently by exploited by means of parallel programming models. In fact, these architectures include support for OpenMP, a well-known parallel programming model from the high-performance domain, in their software development kits.

There has been a significant effort to evaluate the time predictability of the OpenMP tasking model, which enables exploiting fine-grain and highly dynamic parallelism. These works are based on the extraction of a Direct Acyclic Graph (DAG) representing the parallel execution of an OpenMP program [3], and upon which timing and schedulability analyses can be applied for both dynamic [4] and static [5] scheduling approaches. Furthermore, the functional safety of the OpenMP specification has also been analyzed [6].

However, current OpenMP implementations require complex data structures to efficiently orchestrate the parallel execution. For example, in *libgomp* [7], the OpenMP runtime of GNU, these include: (1) a structure per task instance with the code to execute and the data environment; (2) a *hash table* to store the elements in the dependence clauses (i.e., `out` and `inout`), and the list of tasks depending on them, so when a task is created its dependences (i.e., `in` and `out`) are matched against those of the existing tasks; and (3) several linked lists to quickly identify the tasks whose dependences are honored when a task finishes.

Previous works reduce the complexity of the structures in the OpenMP runtime by statically storing the complete *task dependency graph* (TDG) [3]. Nonetheless, the runtime still makes an intensive use of dynamic memory due to: (a) dynamically capturing the data environment, and (b) the task creation policy (which can produce a number of task structures proportional to the total number of tasks of the application).

We have developed an OpenMP framework that pushes the creation of dynamic task data structures to the initialization phase, efficiently executes an OpenMP program, and reduces the amount of memory necessary to exploit parallelism. The framework is composed of: (a) a compiler that expands the complete TDG of an OpenMP application and computes the amount of parallelism exposed in the application to statically allocate the data structures needed be the runtime depending on the task creation policy; and (b) a runtime that implements different policies regarding the moment at which tasks are created and initialized.

Our evaluation shows that we provide the same performance speedup than the one provided by the OpenMP framework included in the SDK of the targeted processor architecture, while significantly reducing the memory requirements.

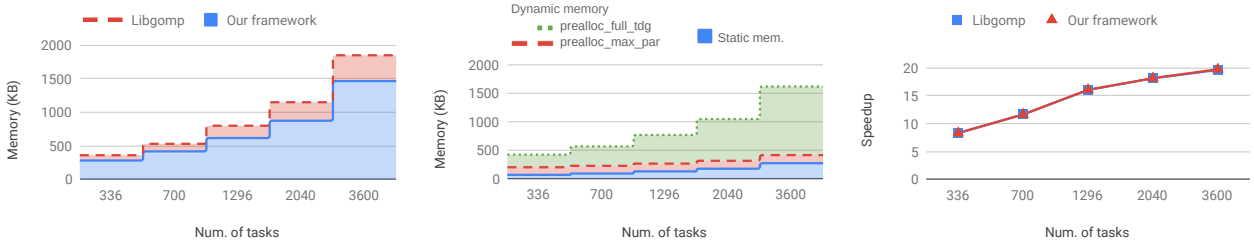
II. Preallocation and Task Creation Policies

With the objective of reducing and bounding the amount of memory needed at runtime, our framework allows different policies with regard to the static allocation of task data structures and the moment at which tasks are created.

Task creation policies. Current OpenMP runtimes usually create tasks as they are encountered. However, these tasks may not be ready for execution due to unmet dependences, in which case they will be using dynamic memory even though they cannot yet be executed. We have implemented a *lazy* policy in libgomp based on a proposal to "un-inline" tasks in a parallel Lisp system [8]. In our case, tasks are not un-inlined, but their creation is pushed until the task is ready to be executed. This means that the amount of on-the-fly memory used to manage task structures is bound by the maximum amount of parallelism exposed in the TDG.

Preallocation policies. We have implemented a series of analysis and optimizations in the Mercurium compiler to expand the complete TDG of an OpenMP application. This enables the static allocation of the structures needed to manage tasks at runtime following two different approaches:

- Allocate as many structures as tasks in the TDG. This reduces the use of dynamic memory, although the total amount of memory needed remains the same.
- Allocate as many structures as parallelism is exposed in the TDG by using a Matlab based algorithm that computes the maximum degree of parallelism of the TDG. This drastically reduces the total amount of memory required by the application.



(a) Total memory consumption (b) Preallocation memory consumption (c) Performance Speedup with 24 cores

Fig. 1: Runtime evaluation for the HoG application depending on the number of tasks.

III. Evaluation

A. Experimental Setup

Processor Architecture. Intel Xeon Platinum 8160 [9] featuring 24 cores.

Framework. GCC 7.3, implementing OpenMP4.5 [10].

Application. A pedestrian detector from the real-time embedded domain based on the computation of histogram of oriented gradients (HoG), available in the VLFeat library.

B. Performance speedup

Figure 1c shows the performance of the native OpenMP framework (labeled as libgomp) and our framework preallocating only the required amount of parallelism and using lazy tasks creation, for the HoG application. The speedup of the two versions is almost the same when using 24 cores (considering the average time of 50 executions with the same data input).

C. Memory usage

Figure 1a shows the maximum memory space (in KBs) required to execute the HoG application with libgomp, compared with our framework without preallocation or lazy tasks. Here, our framekork reduces the total amount of memory used due to the static management of task dependences, which is more memory efficient than the libgomp hash table implementation.

The lazy tasks policy allows preallocating only as many task structures as parallelism is exposed in the TDG. Figure 1b shows the memory consumption of the HoG application with our framework, for several numbers of tasks. The lazy task creation policy reduces drastically the use of dynamic memory, compared to the full allocation of the TDG. Also, the static memory usage increments with the number of tasks, as task dependences are stored statically by the compiler.

Our preallocation mechanism calls a runtime routine to do the allocation at initialization phase. In the future, we will use static structures, eliminating the use of dynamic memory.

IV. Conclusion

Despite the proven benefits of OpenMP to develop critical real-time applications, current implementations are not suitable due to (a) the potentially high amount of memory used to hold tasks, and (b) the intensive use of dynamic memory to allocate task data structures. This limits the use of OpenMP in systems where memory is restricted, and also jeopardizes the qualification for critical real-time systems.

Our work allows limiting the amount of memory required at runtime to handle on-the-fly tasks while showing the same performance as the original runtime. Furthermore, our framework can statically allocate the data structures, so the amount of dynamic memory used by the runtime can be drastically reduced, hence shortening the gap towards the qualification of OpenMP frameworks for critical real-time systems.

V. Acknowledgment

An extended version of this work is to be submitted to CASES 2019. This has been developed under contract N 4000124124/18/NL/CRS (High Performance Parallel Payload Processing for Space) from the European Space Agency.

References

[1] B. D. De Dinechin, D. Van Amstel, M. Poulhiès, and G. Lager, "Time-critical computing on a single-chip massively parallel processor," in *DATE*, 2014.

[2] Texas Instruments, *66AK2Hxx Multicore Keystone II System-on-Chip (SoC)*, 2012. [Online]. Available: www.ti.com/product/66AK2H12

[3] R. E. Vargas, S. Royuela, M. A. Serrano, X. Martorell, and E. Quiñones, "A lightweight openmp4 run-time for embedded systems," in *ASP-DAC*, 2016, pp. 43–49.

[4] M. A. Serrano, A. Melani, R. Vargas, A. Marongiu, M. Bertogna, and E. Quiñones, "Timing characterization of openmp4 tasking model," in *CASES*, 2015, pp. 157–166.

[5] A. Melani, M. A. Serrano, M. Bertogna, I. Cerutti, E. Quiñones, and G. Buttazzo, "A static scheduling approach to enable safety-critical OpenMP applications," in *ASP-DAC*, 2017.

[6] S. Royuela, A. Duran, M. A. Serrano, E. Quiñones, and X. Martorell, "A functional safety openmp* for critical real-time embedded systems," in *IWOMP*, 2017, pp. 231–245.

[7] GNU, "libgomp," 2018. [Online]. Available: gcc.gnu.org/onlinedocs/libgomp/

[8] E. Mohr, D. A. Kranz, and R. H. Halstead, "Lazy task creation: A technique for increasing the granularity of parallel programs," *TPDS*, vol. 2, no. 3, pp. 264–280, 1991.

[9] Intel, "Intel Xeon Platinum 8160," 2018. [Online]. Available: ark.intel.com/products/120501/Intel-Xeon-Platinum-8160-Processor-33M-Cache-2-10- GHz-

[10] OpenMP ARB, "OpenMP 4.5 Specification," 2015. [Online]. Available: www.openmp.org/wp-content/uploads/openmp-4.5.pdf

Adrian Munera received his BSc in Computer Engineering (Computer Architecture specialization) from Universitat Politècnica of Valencia (UPV) in 2018. He is finishing his MSc in Innovation and Research in Informatics at Universitat Politècnica de Catalunya (UPC), developing his thesis in the field of OpenMP and real-time systems. He is also working as a research student at BSC in the CAOS (Computer Architecture - Operating Systems) group.